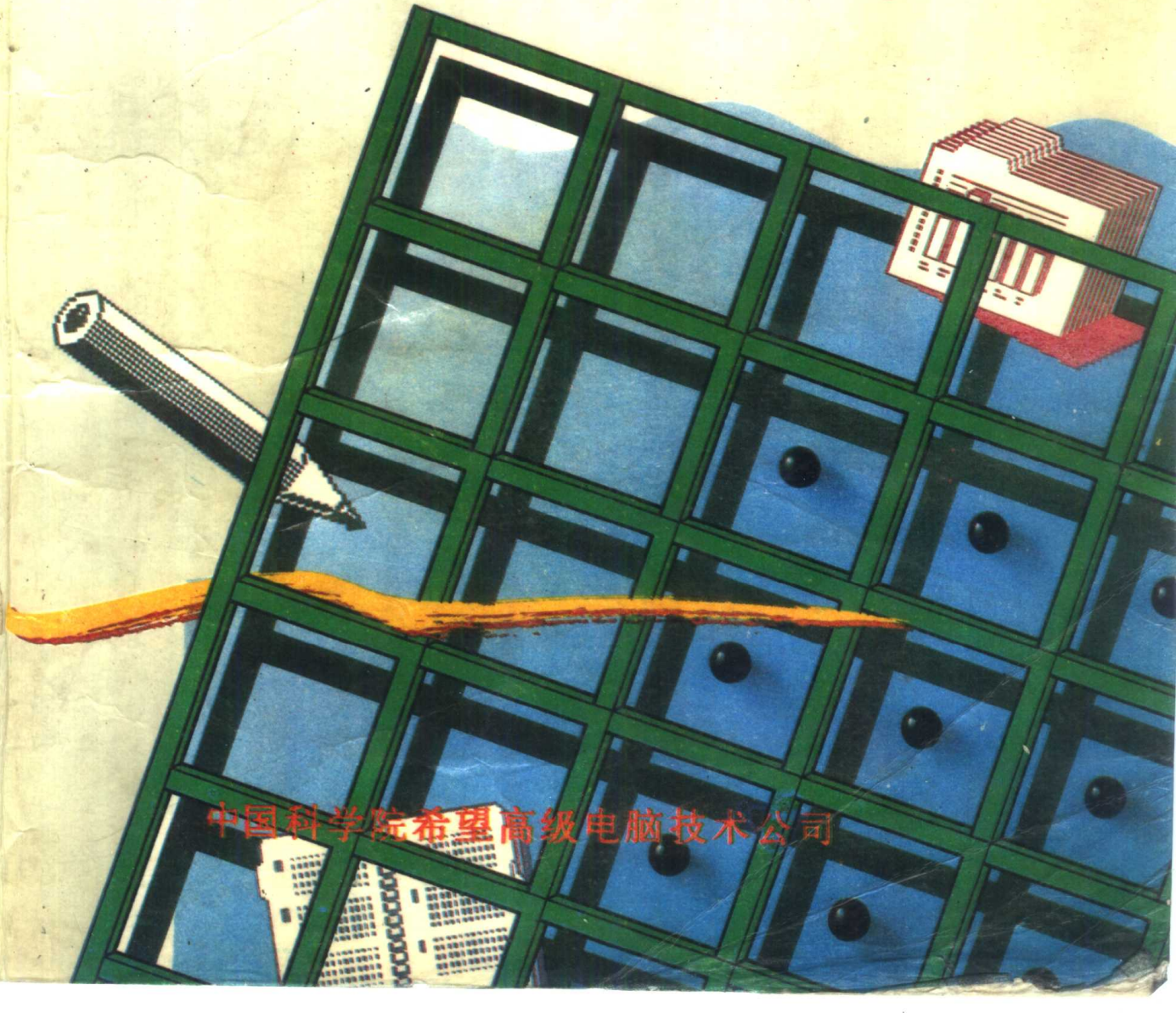
用C语言开发微机用户接口技术

显示和多窗口技术·由顶向下设计方法·数据隐藏技术·模块程设技术和多种程设技巧

王玉书 编译



用C语言开发微机用户接口技术

王玉书 编 译

中国科学院希望高级电脑技术公司

一九九二年二月

内 容 提 要

本书介绍窗口软件的设计技术与实现，尤其提供了完整的实现代码。全书为七章。第1章和第2章介绍了计算机图形学的基本概念、术语和程序设计的环境，第3章和第4章介绍了与物理显示屏和键盘接口，第5章给出了一个编辑实例，第6章介绍了窗口设计与实现。第7章介绍了虚拟显示屏的设计与实现。

本书可供作为应用软件开发研制人员，计算机软件专业的教师和学生的参考书。

用C语言开发微机用户接口技术

编 译 王玉书

编 辑 玉 兰 人 华

北京市新闻出版局

准印证号：3061-90061

订购单位：北京8721信箱资料部

邮 码：100080

电 话：2562329、2567387

乘 车：320、302、332路车至海淀黄庄下车

办公地点：希望公司大楼一楼往里走101房间

■ 北京市新闻出版局
准印证号：3061－90061
■ 订购单位：北京8721信箱资料部
■ 邮　　码：100080
■ 电　　话：2562329
■ 传　　真：01－2561057
■ 乘　　车：320、332、302路
车至海淀黄庄下车
■ 办公地点：希望公司大楼一楼
往里走101房间

译者前言

近几年来，用户接口的设计越来越受到人们的重视。这是因为提高用户的工作效率已经成了计算机（特别是微型机）应用软件的重要研究课题。求助信息(help)、菜单、提示和多窗口屏幕等技术给用户开辟了非常方便的工作环境，尤其是多窗口技术，它不仅方便了用户，同时也提高了计算机的工作能力。

本书为开发微机的用户接口提供了完整的理论描述和具体实现的C程序。书中给出的虚拟键盘、虚拟显示屏和多窗口（给出的例子最多可建立25个窗口，如果需要更多，可自行设置更大的窗口和数目）等技术是研究开发用户接口以及用户接口管理系统（UIMS）的极好材料。

书中给出的编码适用于目前广泛流行的两种微机操作系统——UNIX和MS-DOS。

本书的价值不限于显示技术，同时也涉及由顶向下的设计方法、数据隐藏技术模块程序设计技术、可移植程序的设计和C语言程序设计方法等多种程序设计技巧。对应用软件研制人员、计算机软件课程的教师和学习C语言程序设计的人员是一本很好的学习和参考书。

序　言

我写这本书的目的是为了帮助你设计借助字符显示器及键盘与用户对话的C程序。这些到处可见的设备大体上分两类：一类是ASCII字母数字型的分时终端；一类是带有一体化显示屏的个人计算机。为这些设备编制程序并不困难，我想大多数读者已做过这类工作。但是在为这些设备编程时要实现模块化，可移植，尤其是快速度，却是困难的。

本书中我所涉及的技术及算法对如下两种广泛应用的编程环境是特别适用的：所有版本的UNIX系统（AT&T，Xenix，Berkeley），和IBM PC类计算机的MS-DOS系统。本书有关于UNIX的Curses和Termcap两个软件包的详细信息，以及一些帮助你在不同的情况下进行选择的指导。IBM PC有极快的显示屏，我这里提供了使用它所需要的所有信息。在1982年，我完成了早期IBM PC窗口操作产品之一，EDIX屏幕编辑，它因速度快而受注目。从那以后我学到了很多东西——本书中的屏幕编辑比EDIX还要快！

本书是按抽象层次组织的，因此最好是从头顺序往下读，当然首次翻阅时你可以跳过大部分实现的细节。本书由底向上展开。首先从能被显示的抽象对象字符以及它们的属性开始。然后介绍物理屏幕，即字符矩阵，以及一个通用接口，该接口有五种不同的实现：直接的终端换码序列，Termcap，Curses，IBM，PC基本I/O系统（IBM PC Basic I/O System），以及IBM PC内存映象显示。接下来是物理键盘和虚拟键盘——它可以实现所有的功能键。下一层是窗口，它使一个物理显示屏能被分成几个较小的，可重迭的矩形区域。最后介绍虚拟显示屏，它允许一个窗口后的文本大小任意，甚至比物理屏幕还大，而且能自动卷动。把这些东西加以组装就成了屏幕编辑程序。首先它使用物理显示屏。对它进行重新编程后使它应用窗口，再改写，使用虚拟屏幕。

本书给出全部编码，包括直接处理IBM PC硬件的最低级的汇编语言程序。我曾想仅提供那些精彩的部分而把细节作为练习，但我没有那么做，因为我自己喜欢读完整无缺的程序，也许你也如此吧。况且，那些我认为没啥大价值的部分或许正是你想要读的部分。然而这并不意味着要求你去读所有的编码——差不多九千行！你可能打算找出我在主要章节中强调的一些东西，并且参照阅读你感兴趣的那部分编码。我希望你能在你的当地系统上运行这些程序。任何版本的UNIX和任何IBM PC兼容机都可以。附录C列出什么地方可得到计算机可读程序。

我曾假定你熟悉C，已达到利用C写过一些不简单的程序的程度。如果你有点生疏了，应将C的书放在手边。你还应懂一些你要使用的操作系统，UNIX或者MS-DOS哪种都行。如果你已在该环境下用C编过程序就好了。你不必了解显示器和键盘的任何东西。

本书能适应抱有不同目的的读者的需要。对于UNIX程序员来说，它是一本关于如何使用Curses和Termcap以及如何使用有关控制和读取键盘的系统调用的书。对于IBM PC程序员来说，它是一本关于如何在BIOS和显示一存贮级进行编程的书，包括处理彩色监视器的困难问题。对于学习C的人员来说，它是一本关于模块化编程的书，包括了实际的有实用价值的程序——可不是为了凑成一本教科书而写的那种不实用的程序。对于想学习有关窗口的读者来说，这本书对窗口系统的实现方法作了比较并给出一种实现的细节。最后，对于那些想得到某些实际显示器及键盘编码的读者来说，它就不是一本书而是一个软件包了！

当我开始写这本书时我知道，把这本书的着眼点放在字符上而不是图形上会引起一些人的惊讶。我的朋友们总是问我是否也拿出图形方面的书；我想他们是怀疑光谈字符怎么能写一本书呢？当然了，图形——特别是带有窗口，鼠标器等等的图形问题应是本书“内含”的事情了。不过把精力集中在字符上，我有我的原因：

1.对于廉价终端和计算机来说，字符显示要比图形显示快得多，在第1章中将解释清楚这一点。我发现，我能设计较好的使用字符的用户接口，因为我能得心应手地使用多重窗口，help屏幕，多行错误信息，下落式菜单，和四向卷动——所有这些都不会怠慢用户。图形显示器是很吸引人的，但是除了速度极快的计算机，在其它的计算机上动作都太慢，这一点使它难以启用。

2.大部分事务管理系统使用联接到多用户计算机上的终端。在这样的终端上做图形是不可行的，因为传输速率不够快而且图形终端造价可能太高。仅使用字符的程序有较大的可移植性，进而就有较多的商业价值。不幸的是，就我所知，这种商业价值有的已不存在了，因为许多事务管理系统上的用户接口深不可测。可能是它们的设计者认为你需要图形显示来把事情做得更漂亮，我可不相信能做到这一点，我希望本书提供的技术能给程序员们提供更好地使用他们现有设备的方法。

3.因为字符显示易于理解，本书的程序比较易读，如果使用图形就不会如此了。（我说的是“真正”的程序，而不是看着玩的）。例如。我可以让你看窗口是如何工作的，使用的是实际的，可运行的编码。在解释算法的时候，我已经包括了关于实现图形的相应方法的注释，这样你也会增进对图形显示的了解。

4.最后，几乎每个程序员都用过带有字符显示器的计算机，因此你可以实验这些程序、还可能丰富它们，如果你有这个想法的话。不需要特殊的设备和软件。如果学生们能使用UNIX终端或PC机的话，本书还可作为C语言，操作系统，或用户接口设计等课程的补充教材。

Marc J. Rochkind

目　录

第一章　基本概念

1.1 浏览计算机图形学

按照〔New 79〕，计算机图形学是“用计算机辅助生成和处理图形的学科”。这个定义当然相当好，由于来自计算机可见的每样东西，即使文本，都认为是一张“图”。并且，任何输入——无论何种形式——以及它的输出都包括在这个图定义里。我们强加交互式的这种额外要求，能缩小够定义的范围。这意味着，当图形正产生时，存在观察图形的人用户，并且人用户控制图形，相对比，被动式的计算机图形学要求由绘图仪输出，或向计算机生成的胶片上输出，该用户只能看着输出，输出完之后，用户才能影响图。

术语交互式提出了用户与计算机间接口的另外一些属性：用户参与这样经常，以至于看起来如会话一样。的确，会话是全部的中心，而图象处理不是全部的中心。必须相当快地生成图象的输出，使得能跟上运行的程序；棋赛能像真的一样，不像一场函邮棋赛。最后，较小的显示区域——不到一平方英呎——应不断地重新使用和有选择地更新。在一卷纸上的图，无论多么快，也不能胜任交互式。

通常，当用户正坐在计算机能在几分之一秒内更新的矩形显示器前面时，发生交互的计算机图形。该用户敲键盘，并且控制某类定点设备，常常是一个小盒，其下有个滚动的球，称之鼠标器（它的连线是它的尾巴）。还能有许多其它的设备，范围从运动场的记分板到围绕着用户的三维电影屏幕，但最常见的一个会满足这本书的要求。

许多交互式图形显示器能画出任意形状的图形，包括安排到方阵里的文本字符，其中每个字符具有同样的高度和宽度。当这就是我们在显示器上所能做的一切时，或者说，我们所关心的一切时，我们称该显示器为字符显示器，另一个广泛用的术语是字母数字显示器。

本书中处理的唯一输入设备是键盘，这样我们所关心的计算机图形学分支可用短语“交互式字符显示器和键盘”来描述。这是计算机图形学的一个小角落，但是对于压倒的多数计算机用户，它是他们经历过的唯一角落。差不多95%以上的计算机应用是基于字符，并且靠键盘取得他们的所有输入。随着硬件变得更好更便宜，软件变得更丰富，计算机图形学和其它领域正迅速地发展。但是，字符和键盘还将会统治很长一段时间。

用户实际接触的由一个显示器和一个或多个输入设备组成的这套装置，我们将称之为工作站。我们将应用这个术语于所有这样的装置，不单单指大的图形显示器和能力强的CPU组成的装置。另外，我们能使用终端一词，但其只意味着靠通讯线与主CPU有联接。我们想有一个术语，其包括集装在计算机其它部分里的显示器和输入设备（如典型的个人计算机）。

现在，无论工作站只显示字符或更一般的图形，主要地取决于如何使用它，不取决于它的内部设计。例如，典型的字符计算机内的硬件能够轻而易举地处理任意图形，只是该终端的CPU没有被编程处理图形而已。的确，若使它能成为图形终端，也可能必须改变其它一些

硬件，如加些存贮器或加速CPU，但是基本设计保持不动。解释说明把一个显示器称为字符的还是图形的困难性的另一个例子是IBM PC，它既可以是字符的，也可以是图形的，这取决于显示软件选择那种模式。再一个例子是Apple Macintosh，它无疑地是图形计算机，适当地运行一个通讯程序，可使它成为一个基于字符的VT-100终端。当它起这样的作用时，不能再认为它是图形显示器。

因此，我们应记住字符只是图形的一个特殊用法，大体说来，不是另外一门技术。好了，那么为什么写只处理字符的程序？有三个理由：

1. 基于字符的程序比图形程序简单得多。因为它们处理少数几种类型的对象（字母，数字和标点符号），它们的大小相同，能用有限种方法对它们定位（比如说，在25×80网格任意处）。

2. 基于字符的程序使用较少的内存，并且更新显示快一些，因为发送到显示器的数据非常少。例如，25×80字符显示器只要输入2000个字节就能填满；用图形映象填满同一区域需30000多个字节[1]。由于响应时间对交互式程序使用的便利性有非常大的影响，往往基于字符的设计更有用。图形接口用起来舒服，但是它们常常太缓慢。另一个想不到的权衡事宜是颜色。在许多IBM PC显示器上，彩色只在字符模式下显示出来——在高分辨率的图形模式下，只显示黑白色。

3. 由于实际上所有交互式显示器都能显示字符，而较少的能显示图形，基于字符的程序能用在任何地方。这种通用性对用户和卖主都有利。

在引言这一章的余下部分，我们不想不必要地区分字符和图形，但是本书余下部分仅处理字符。

1.2 工作站硬件的具体视图

这一节从硬件本身出发，在非常低的层次上考察交互式工作站硬件。当然，硬件的性质极大地影响程序处理它的方法。但这样低层的视图太混乱，以至于不能作为程序设计可行的基础。例如，在硬件一级上，CRT和LCD显示器完全不同，但是使用阴极射线和使用液晶与程序设计员毫无关系。可能有关的是它们的尺寸，分辨率，速度和显示颜色的能力。在第1.3节，我们以这种更抽象的观点考察硬件。在〔Hea86〕有一篇非常好的最新的关于图形硬件的概述，下面的许多内容以此为基础。另一推荐的参考资料是〔Art85〕。

1.2.1 输出设备

交互式图形输出设备由显示屏，驱动它的控制器，存贮当前映象的存贮器，和对计算机系统其它部分的接口组成。我们将分开讨论这四个部分。

当今最常使用的显示屏是阴极射线管，按它的第一个大写英文字母称之CRT。在显象管前部的玻璃内表面上涂有荧光粉。当电子束击在显示屏上一点时，荧光粉发亮光，但不能保持长时间。为保持那个点发亮，电子束要经常击它，比如说，每秒30次。这称作刷新显示器（refreshing）。当常常刷新不足时，显示器闪烁。按一确定的刷新率，某些种类的荧光粉比其它种类的闪烁更严重。这取决于它们的持久性。主要用于字符的显示屏要用长持久性的荧光粉，而用于动画的显示屏（如电视机）使用短持久性的荧光粉。

[1]我的比特映象（位映象）IBM PC显示器是350个象素高720个象素宽。当它在字符模式时，每个字符占有14个象素高和9个象素宽的一个方框。在图形模式填满它需252,000个比特，即31,500个字节。

由于仅有一条电子束，它必须循图移动使得复盖整个显示屏。在光栅扫瞄显示器里，迄今最通用的类型，显示屏组织成由点所构成的行，称之光栅。图象由点组成，每一点称作象素或象元。电子束扫过一行（当它扫过时，线束可有可无），然后下一行，如此下去，在扫过最后一行后，转向顶行。为使电子束从一行转到另一行，关闭电子束再转向显示屏左端，像打字机的回车换行一样。这称之水平回扫。从最后一行回到顶行称之垂直回扫。由于CRT的控制器空间，回扫为CPU提供了时机，CPU可不受干扰地访问显示器的存贮器，当在第3.7节对IBM PC编程时，我们将利用这一点。

点的数目，更精确地说，每线性单位点的数目，称作分辨率。低分辨率光栅垂直约200个象素，水平约160个象素。高分辨率光栅每个方向有1000或1600多个象素，总象素在32,000到1,000,000范围内。对于不太贵的个人计算机，某些典型的显示器对于IBM PC是350×640（总数224,000），对于Apple Macintosh是342×512（总数175,104）。

高分辨率显示器每英吋约有80到100个象素，这取决于象素总数和显示屏的实际尺寸。这看起来非常好，但与便宜质量差的点阵打印机相比，按打印标准来衡量还是太差。激光打印机每英吋有300个点，排字机至1000个点。水平分辨率常常高于垂直分辨率，所以长宽比大于1。当你画图形时，必须考虑到这一点；若不注意，圆和方形变长。

为降低高分辨显示器的成本，通过隔一行一刷新，以正常刷新率的一半进行扫瞄，这称之隔行扫瞄。不幸地，这常引起注目的闪烁。有时所谓高分辨率显示器不得不以低的分辨率操作，以避免隔行扫瞄的闪烁。

除了光栅扫瞄外，只用一条电子束复盖整个显示屏的另一方法是只向要点亮的部分移动电子束（类似绘图仪）。这是随机扫瞄显示。由于它最适于画线，它也称作向量显示器（vector display）。随机扫瞄显示器很少用在现代工作站，因为光栅扫瞄显示器更通用，完全令人满意，又由于许多厂家制造它，价格又便宜。

不同类型的荧光粉发光颜色不同。彩色显示屏在每个象素处有红、绿、兰色荧光粉。每一色有一电子枪，并且阴罩帮助把枪指向该击的点。每个枪的强度都能够改变，可以控制红、绿、兰色的程度，以便产生多种颜色。描述彩色CRT上的每个象素至少需两个比特，并且典型地使用3或4个。因此，彩色需要更多的存贮器和处理。

只操纵电子束就能控制的其它属性不需要特殊的荧光粉。若电子枪以定长时间间隔开闭，象素将闪亮，象素的强度可通过调整电子束的强度来改变。给出由一组象素组成的图形，如一个字符，通过翻转象素的开/关状态，可以以黑白颠倒的方式显示它（视频翻转）。

除了重复地用电子束刷新保持涂荧光粉的显示屏发光外，还有另一种方法，给它衬以可持电荷的栅格。用电子枪在此栅格上把图象一次画出，然后它控制二次的，不断的“泛”电子流。该电子流只激发有图象处理的荧光粉。这称之直象存贮管，即DVST。它能保持非常大的而复杂的图形。DVST不能显示彩色，即使这在技术上不是不可能的。它的最大缺点是不能有选择性地擦去某部分——整个显示器必须同时加以清除。按我们要谈的交互式，它不能用作交互式图形显示器。DVST是随机扫瞄，不是光栅扫瞄的。

除了CRT外，具有类似性质的技术用在特殊情况里。等离子板平而坚固，所以倘若有足够的电力可利用，对于袖珍计算机它是理想的。当无足够电力时，液晶显示器（LCD）有时

---

①在具有25×80字符LCD光屏的电池电源“laptop”计算机上，我写了这本书的好几章。工作在山区公园的野餐桌上，就解决了照明问题。

能用，但必须有大量外界光，否则几乎读不了显示器②。这个问题可由自带辅助光照射显示器加以解决，但它需要辅助电源。

由于刷新的CRT必须经常刷新，以至于由应用CPU直接地操纵电子束不现实。而是由专门的控制器操纵它。实际上，控制器可与CPU装在一个箱体里（如在IBM PC上），或者控制器与显示屏装在一个箱体里（如在终端里）。

控制器也许非常简单，仅仅充分灵活地使电子枪瞄准，并且在适当的时候开关电子枪，也许是一台拥有自主权的功能很强的计算机，它有自己的微处理机和存贮器。事实上，好最把主CPU和显示器看作分布式处理系统的两部分。

简单控制器只能显示象素，或只能显示固定大小的字符，或者两者都能显示。复杂的控制器也能画出几何图形，如圆和线，某些能以各种字体和大小显示字符，某些已实现的计算机图形接口（CGI），其不久将成为ANSI标准〔CGI85〕，某些提供窗口系统。某些控制器供填空表格的。有一种想法是把控制器联到更专用的CPU上，减轻主CPU的负担。

控制器保存它正显示的数据在当地存贮器里，以免影响应用CPU。若这种存贮器的组织类似于光栅，它称作帧缓冲区。在较高档的控制器里，其存贮可能组织成显示表，它可以具有控制器的程序设计员所喜爱的任何结构，但它必须可灵活地加以修改和快速地被访问，一个控制器可以既有显示表又有帧缓冲区。

应用CPU必须与控制器通讯。在终端里，每一控制器联于一个通讯口，并且它们通过硬联线，网络，或用解调器和电话线等进行通讯。有许多不同的通讯协议可用：RS-232C，SDLC，X.25等等。

终端的最大缺点是填满屏幕需时长，这严重地限制了能用的用户接口设计技术。下落（drop-down）菜单和弹出（POP-UP）对话柜是不实际的。横向卷动（或称滚动）——用于表格——事实上不可能做到。把显示器和主CPU更紧密地联接起来更好些。在个人计算机里，使它们联接起来最见常的方法是使它们共享帧缓冲区。应用程序在帧缓冲区写的东西几乎瞬时间就能显示出来。很显然，在这种情况下，应用程序必须知道帧缓冲区。典型地，它是字符和属性的矩阵或位图（bitmap）。

让我们快一些描述本书所用的两显示系统：IBM PC显示器和典型的ASCII终端。

IBM对PC提供了几种显示器供挑选，其它卖主提供得更多。每一个都有显示文本的能力（25行×80列），并且大多数能以不同的分辨率显示图形。最通常的配置：

1. IBM单色显示适配器（MDA），它只显示黑白字符——完全不显示图形。字符以9×14大框画出，而且它们特别清楚。一个字节的所有8位都用，所以能画出256个字符，除ASCII字符外，还有外国字符，画线符和数字符号以及怪模样的字符，如笑脸和音符。每一字符与一属性字节相结合。属性可以是闪烁，黑体（重体），反色，加下划线或正常的，这些属性可以单独地使用也可以结合地使用。该字的一些位没有用，这是由于这一存贮安排是用于彩色的，不是专用于黑白色的。该显示控制器和主CPU共享帧缓冲区（它是存贮映像的）；每一个能在任何时刻访问它，并不受干扰。显示器以112,000波特率运行，几乎比1200波特率的拨号终端快100倍。你必须用一种专门的显示器，通常称TTL显示器，操纵该控制器。

2. IBM彩色图形适配器（CGA）。在字符模式下，它类类于单色适配器：它使用同

单色显示适配器同样的存贮安排，支持除加下划线外的所有属性。在属性字节里表示下划线的那一位和对于单色没有用的那些位用于彩色。有16种前景色和8种背景色。在图形模式下，对分辨率可做选择：200象素高320象素宽或640象素宽。在低分辨率下，一象素可以是四色中的一个；在高分辨率下，显示黑白色。

在字符模式和高分辨率模式时，主CPU和显示控制器不能同时访问帧缓冲区，于是除了回扫期间外，CPU不能访问帧缓冲区。在第3.7.1节将详细讨论这个问题。

你能把各种监视器挂在控制器上，范围从电视机到RGB监视器（所以这样称呼，由于它的枪直接地响应控制器生成的红、绿、兰信号）。监视器甚至可能无色，它只能显示灰色阴影。程序不能确定它配上了哪种监视器——这要问用户——或可以用哪种颜色(或阴影)。几乎没有IBM PC程序很好地处理了这个问题。

当LCD显示器用在IBM兼容机上时，它通常仿真彩色图形适配器。但它只能显示黑白色——甚至不能显示灰影。某些应用程序的输出，在别的显示器上看起来鲜艳夺目，但在LCD上看不到鲜艳的彩色。

3. IBM增强图形适配器（EGA），它能仿真其它适配器（MDA和CGA），另外的好处是CPU能不受干扰地访问帧缓冲区。它也提供另外几种图形模式。模式之一是能在单色适配器所用的显示器上画出图形（350×640象素）。其它模式用于彩色（200×320或640，具有16种色；350×640具有64种色）。对于彩色，EGA使用专门的监视器——可与CGA一块用的标准的RGB监视器不能用。

4. 特强图形适配器（HGA）。像EGA一样，它能在IBM单色监视器上生成图形，只不过以较高分辨率显示它（348×720）。HGA也与单色显示适配器一样能显示文本。它完全不能显示彩色，并且它不能驱动任何其它类型的监视器。

以上列出了四种类型的控制器，还至少有半打其它类型的控制器广泛地使用着，包括BM公司1982年推出的新的480×640控制器。幸运地，与图形不一样，各种适配器对字符显示差别不大。所有的适配器都能显示同样的字符，并且都是25×80。

典型的分时终端有24或25行80列的字符显示器和键盘。该显示器至少能处理ASCII字符集，有时也能处理非标准的外文字符或画线字符。它通过硬线或电话线与应用CPU相连，以全双工方式工作，这指的它能同时接收和发送。一个字符或一组字符，当它一敲出，就由键盘发送出去，而不必等待一行或一页敲完才发送。通常敲入的字符由主操作系统或应用程序处理，并且在敲下一个字符之前或正敲之时，显示一个响应。即使这种处理通讯系统和CPU的处理负担很重，但是得到的灵活性很有价值。

仅有几个显示操作由ASCII字符集标准化：回车，换行，回退，水平制表符（tab），以及其它一些。为允许另外一些操作，如滚动，清除，移动光标等等，换码序列能发送到终端上。这些序列由换码字符（Escape）开始，以便使它们与一般字符序列相区别。跟在换码字符之后的东西，随终端不同而不同，这造成了对用在不同模式下的显示应用程序编程很困难。对这个困难，UNIX Termcap系统提出了一个非常好的解决办法（Berkeley加利福尼亚大学研制的），在第3.4节将详细地描述它。对于换码序列有一个ANSI标准，但遵守这个标准的制造者也实现了一些超过此标准的性能，为此他们编了他们自己的序列。这样，事实上，只有额外的一种终端类型要考虑。

另一通用终端设计由IBM 3270类终端为代表。这些终端使用IBM的EBCDIC字符集，

不是ASCII字符集。设计它们主要用于表格应用。通过把每行看作一个域，来实现文本编辑。这样使用减轻了屏蔽（mask）这一担子。屏蔽用在显示屏上建立数据项域。用户只向这些域里敲，并且只能使用适当种类的字符（字母，数字等等）。光标在屏幕上各处移动纠正错误做局个处理。当表格完成后，用户按enter键，从域中收集数据，编成信息，然后发送到应用CPU。也有些ASCII终端，它们强制向表格的域敲入数据，并且在发送任何东西之前，允许做局个编辑。

几乎所有的终端都以低的通讯速度运行（指PC标准），范围以每秒30至960个字符，当设计在这样的终端上运行的应用程序时，必须要考虑速度。通常，为了使用户感到满意，尽量提供好的人的接口，如显示菜单，对话框和窗口等等。这是为什么分时系统的老用户喜欢命令驱动接口的一个原因。在第3章我们将看到终端的速度，如何使与它们的接口复杂化，并且我们将探索一些有用的程序设计的技巧。

有图形终端，还有具有图形选择的字符终端。发送位图到图形终端不实际，因为花时间太长，例如，以每秒960个字节的速度，要填满350×720个象素的显示器屏幕约30秒。较好的方法是在较高的一级上，比如说，使用计算机图形接口，使得显示器与终端相连接。它让应用程序发送线、多边形、图、文本、以及其它大的对象，不仅仅是位（bits）。再更好点，安装微处理机和一些存贮器在终端里，使它自己能进行大部分的图形处理。主要是关于图形的那些应用，如计算机辅助设计（CAD），几乎完全能在终端上运行。显然，那它不再是终端——它是无磁盘的个人计算机。

本书中我们关心的是字符终端，并且只是全双工的，ASCII的一次一个字符的这一种终端。

1.2.2 输入设备

大多数交互式设备允许你按按钮或用定点器定在某一东西上，或在显示屏上或在工作站旁边的文件上。最常见的按钮设备是键盘，典型地它约有85个按钮，以及更多的组合按钮。在鼠标器上至少有一个按钮。声音输入也象按钮输入一样，因为它用于输入文本或命令。定点器（pointing device）样式很多。它们包括鼠标器、控制杆、跟踪球、光笔、图形板、触感显示屏，还有小装置，你能把这小装置放在你的前额上，并像矿灯一样地瞄准。

几乎所有的键盘使它们的大多数按键安排成标准的布局，通常称作QWERTY，即使有时字母和Dvorak布局也采用。对于输入文本，键的布局不影响应用。然而，当控制键组合起来用于模拟功能键时，键功能的指定与具体布局密切有关。例如，Control-E，Control-X，Control-S和Control-D在键盘上形成一个钻石形，所以把它们用作光标键。

所有键盘除了类似打印机的部分外，还有另外一些键用于光标移动(典型地标以箭头)，编辑（如Insert，Delete）和一般功能（如F1，Attension）。由于这些键差不多都没有包括在ASCII标准里，并且由于这些键的结合那么多，都按7位码处理它们是不可能的，某些键盘为了功能键使用了全部八位（给出另外的127个编码），但这会产生通讯困难，因为最高位常常被操作系统，设备驱动器，通讯处理器和网络修改。因此对于功能键，发送一个换码序列更一般。于是应用程序必须处理这一情况，即单次键击可产生任意多个输入字符。

对于键盘的功能键还没有标准化。结果，使得难于设计好的用户接口。好的用户接口意味着利用这些加标号的键，并且对于键盘在一定程度上可移植的。在第4章，我们对此问题

给出了一个简洁的解法。一个更难办的问题（对于它我们给不出一个解决办法）是键盘上的键随用户而改变时，难于写出用户手册和help屏幕。通常用两种方法：或者一般地借助于键（如Home键），并且认为用户已记住了这些一般键的约束含义；或者除了ASCII键外，其它键均不考虑。也就是说，不管终端有没有向上箭头键都使用Control-E。

定点器，如鼠标器，正常地被认为是图形设备，但它也能与字符显示器一起使用。它们一般地与显示屏上的光标相连接，而不是通过实际移动这种小器具来行走。CPU能感知光标的位置（按行和列坐标），并且如果该装置有按钮的话，CPU也能知道当前按下了哪个钮。由于光标总有个位置，应用程序要花时间不断地取到它的位置。于是，通常有一个设备驱动器，它监视该器具并且只当请求它时，提供输出。终端很少有定点器，即使没有什么技术上的问题。这可能由于现有的分时应用程序很少使用鼠标器，因为买主方面对于定点器几乎没有什么要求。

即使定点装置广泛地应用于基于字符的应用程序里，主要由于篇幅的限制，我们在这本书里完全不处理它。若光标定位的话，如第5章文本编辑中的，我们采用按光标按钮的方法。

1.3. 工作站硬件的抽象视图

当工作站硬件交付于程序设计时，其细节含混不清令人感到迷惑的话，谈论工作站硬件是笑话。对于程序设计，我们要有一个视图，其较少与物理性质有关。我们通常不关心一个功能在何处执行的——在控制器、在操作系统里、或在程序里——而关心提供了什么样的接口。对于程序设计员显示器作用像一个8比特字符数组，而不是1比特的象素。但它很少与由象素形成字符的方法有关，即不管字符形成在ROM里或在RAM存贮器里。一个相当抽象的视图得到的不只是兼容性和可维护性——使我们的头脑清晰。

一个抽象显示器能显示一组可见对象，每一对象在一个指定的位置，并且每一个具有一些相结合的属性，如强度、颜色等。对于我们最关心的这些显示器，位置总以直角坐标系表示，因此每一位置由行号和列号表示。显示器最顶行行号为零，向下增加；最左列列号为零，向右增加。行数和列数确定了用于说明对象的位置数目，即分辨率。

现在我们能按所显示的对象区分图形和字符显示器。为获得最大灵活性，把一个对象看作是一个象素。这样的显示器称作位映象（bitmap）显示器。但记住若象素除具有存在与不存在属性外，还有其它属性，需要几个比特描述它。象所有的其它对象都有象素构成一样，这种类型的显示器是其它类型的超子集。问题是这样多的象素构成的对一个对象的描述不易使用：消耗时间计算和转换，消耗时间从应用CPU传送到显示控制器。

若我们愿意用灵活性换取紧凑性，能选择更多有意义的对象。由于许多图象的元素是——或可能是——线、圆、多边形和其它容易说明的几何图形，处理这些类型对象的显示器能够非常容易非常快地处理它们。例如，圆可用三个整数描述，两个用于圆心一个用于半径，总用48个比特。对于同样的圆用位映象说明需1000个比特。若标准的几何形状不适合的话，显示控制器能设计成处理各种形状的图形，如电阻器、办公椅、眉毛飞船等任何东西。这些图形在处理开始时，化为位映像，并且永久地存在控制器里。通过基本图号码识别系统，能把基本图作为图象加以访问。该控制器能够放大、旋转、和着色所有的对象。

字符显示器限制对象为一组预先定义的字符形，它们或者是内带不可改变的，或者显示

器初始化时化为位映象的。最一般的例子是具有256种可能字符的显示器，使得一个字节足够说明一个字符。但是，例如对于汉字显示器，可以用两个或多个字节。进一步的限制是字符不能随便显示在任何处，只显示在设计好的单元处，这些单元排列成矩形数组。一个单元可以是8×8的方形、7×9或7×14，或者设计员所喜爱的任何方形。但若行和列解释成字符位置而不是象素位置时，这不影响显示器的程序设计。因为我们把字符显示器看作例如25×80字符的，而不是看做200×640象素的这种抽象使得我们的程序正常地工作，而不管字符是如何画出的。

在知道显示的一组对象是一个接一个的情况下，不是给出每一个对象的位置，应用程序只说明该组对象的第一个对象的位置，这样能工作快一些。进一步，一行末尾的对象看作与下一行的第一个对象相邻。因而对于满屏幕的对象，简单地提供一个对象标识符的数组，而不必给出行号和列号，就可以说明它。当与显示器的连接是通过共享帧缓冲区时，就是这种情况。当连接是通过通讯线（即，显示器是一个终端）时，更一般的是对于每一对象，说明行号和列号，或者对于发被的每组邻接对象，至少说明一个行号和列号。

隐藏应用程序和显示器间连接的物理属性使应用程序不知道是否它使用共享存贮、直接存贮访问（DMA）、网络、硬线还是电话线接口等，这是一个好想法。的确，应用程序编完以后很长时间，后来安装者才能做出决定。所用的最好的抽象，特别对于C程序，是直接地函数调用，它看起来这样：

```
displayobjects (row, column, objectcount, objectlist, attributelist)
int row, column, objectcount;
OBJECT *objectlist;
ATTRIBUTE *attribute;
```

变量objectlist和attributelist是抽象类型的数组，它的定义留给读者去想象。调用这个函数能显示任何对象，无论象素、字符或棋子。你将看到此模型函数和函数PSwrite、Wwrite、以及VSwrite之间的相似性，以后我们将介绍这些函数。

即使在抽象一级，程序设计员也需要了解显示器的一些性质。显示一个东西，它需要多长时间？这对用户接口设计有非常大的影响。对象可读吗？比如说，这影响是否用户在显示屏上校读一个文件或是否必须打印出它。能显示多少个对象？显示器的尺寸当然是重要的。无论什么性质是有关的，应以抽象的方式叙述它们，这样程序员就不被迫处理这些低层性质的东西，如电子枪回扫区间，LCD对比度，字符单元的大小等等。

就输入一面而言，另一组抽象能帮助程序设计员不接触令人混淆的细节。在键盘上每按一次键产生一个键击，用一个整数代码标识它。小于128的键代码用于ASCII码，它已为人们所熟知，并且已经标准化，直接地用于程序里。大于128的代码指定为抽象功能键，使用一种方案，类似这样：

```
Cursor up        301
Cursor down      302
Insert           312
  ...
```

现在该程序能写成接收键代码序列，并且映射它成实际键盘的问题能单独地处理。这个虚拟键是第4章的课题。